Status of the Claims

1-26. (canceled).

27. (currently amended) A computer software product, comprising a computer-readable non-transitory medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to execute a method for validating a multi-processor design by simulating program execution, comprising the steps of:

identifying a ~~resource, comprising~~ plurality of mutually dependent non-adjacent resources having non-contiguous addresses accessed by a multi-processor architecture test program that includes a first simulated process and a second simulated process, wherein each of the mutually dependent non-adjacent resources has an adjacent resource at a contiguous address therewith;

identifying a set of value-lists, each value-list comprising a list of elements, wherein each of the elements corresponds respectively to a permissible value of one of the mutually dependent non-adjacent resources and to a permissible value of the adjacent resource thereof ~~containing permissible values of said resource~~;

associating a set of non-unique values for said mutually dependent non-adjacent resources ~~resource~~;

initializing said mutually dependent non-adjacent resources;

using said initialized mutually dependent non-adjacent resources executing said test program by the steps of:

executing a first sequence of instructions in said first simulated process; and

while performing said step of executing said first sequence, executing a second sequence of instructions in said second simulated process;

wherein each of said ~~resource is~~ mutually dependent non-adjacent resources is accessed by at least one of said first simulated process and said second simulated process, and wherein upon completion of said steps of executing said first sequence and executing said second sequence, a member of said set of non-unique ~~nonunique~~ values is required to be present in ~~said resource~~ each of said mutually dependent non-adjacent resources and in said adjacent resource thereof;

creating, by the computer executing said test program, a set of tagged ~~value-list~~ value-lists, each tagged value-list identifying an allowed subcombination of values of one of the mutually dependent non-adjacent resources and the adjacent resource thereof resulting from the steps of executing said first sequence and said second sequence, and formed by tagging members of a list of predicted results with a combination identifier which includes a string of literals identifying a particular outcome of said test program, wherein at least a portion of said mutually dependent non-adjacent resources have a common combination identifier, each of said tagged value-lists ~~value-list~~ comprising:

said ~~set of non-unique~~ list of elements ~~values~~; and

said combination identifier;

~~replacing said set of value-lists by said set of tagged value-lists, such that said set of tagged value-lists is used in place of said set of value-lists;~~

upon completion of an execution of said first sequence and said second sequence comparing contents of said mutually dependent non-adjacent resources and said adjacent resource thereof with a

corresponding member of said set of tagged value-lists to match the allowed subcombination of values thereof, said step of comparing performed by selecting commonly tagged members of said set of tagged value-lists for each said combination identifier; and

responsively to a failure to match any allowed subcombination of values in iterations of said step of comparing reporting that an architectural violation of the multi-processor design exists

~~validating the processor design if a content of said resource is equal to a member of one of said set of tagged value-lists~~.

28. (original) The computer software product according to claim 27, wherein said resource is a memory resource.

29. (original) The computer software product according to claim 27, wherein said resource is a register.

30. (previously presented) The computer software product according to claim 27, wherein said set of non-unique values is said set of value-lists.

31. (currently amended) The computer software product according to claim 30, wherein said adjacent resource comprises a first adjacent resource and a second adjacent resource having a contiguous address with the first adjacent resource, and each member of said set of value-lists ~~lists of values~~ comprises a first value and a second value, said first value being a permissible value of said first adjacent resource, and said second value being a permissible ~~values~~ value of said second adjacent resource, and said step of verifying further comprising the steps of:

identifying a valid member of said set of value-lists ~~lists of values~~, by the steps of:

verifying an equality between a content of said first adjacent resource and said first value of said valid member; and

verifying an equality between a content of said second adjacent resource and said second value of said valid member.

32. (currently amended) The computer software product according to claim 27, wherein said ~~resource~~ mutually dependent non-adjacent resources comprise ~~comprises~~ a first resource and a second resource and said set of non-unique values comprises a first set of non-unique values that is associated with said first resource, and a second set of non-unique values that is associated with said second resource, the method further comprising the steps of:

associating a first member of said first set of non-unique values with a second member of said second set of non-unique values; ~~and~~

~~wherein said step of verifying comprises the steps of:~~

verifying ~~an~~ a first equality between said first resource and said first member; and

verifying ~~an~~ a second equality between said second resource and said second member.

33. (currently amended) The computer software product according to claim 32, wherein said step of associating said first member is performed by tagging said first member and said second member with a common combination identifier of ~~a~~ the particular outcome of said test program.

34. (currently amended) The computer software product according to claim 32, wherein said first member of said first set of non-unique values comprises a first list of values, and said second member of said second set of non-unique values comprises a second list of values, respective elements of said first list of values being permissible values of said first resource and adjacent resources having contiguous addresses thereof, and respective elements of said second list of values being permissible values of said second resource and adjacent resources having contiguous addresses thereof, wherein said step of verifying a first equality comprises ~~the steps~~ of verifying an equality between a content of said first resource and adjacent resources thereof with corresponding elements of said first list of values; and

verifying a second equality comprises verifying an equality between a content of said second resource and adjacent resources thereof with corresponding elements of said second list of values.

35. (currently amended) The computer software product according to claim 27, wherein said step of associating said mutually dependent non-adjacent resources with said set of non-unique values is performed prior to said step of executing said first sequence of instructions.

36. (currently amended) The computer software product according to claim 35, wherein said step of associating said mutually dependent non-adjacent resources with said set of non-unique values is performed by the steps of:

defining a results section in ~~said~~ input statements, and entering all permissible values assumable by said ~~resource~~ mutually dependent non-adjacent resources and an identifier of said ~~resource~~

mutually dependent non-adjacent resources in ~~an~~ respective entries ~~entry~~ of said results section.

37. (previously presented) The computer software product according to claim 27, wherein said step of executing said test program further comprises the steps of:

generating said first sequence and said second sequence to define generated instructions;

simulating one of said generated instructions in said first simulated process and said second simulated process;

maintaining a store that contains a set of values that are assumable in said resource during said step of simulating said one of said generated instructions; and

thereafter determining whether said store contains non-unique values.

38. (original) The computer software product according to claim 37, wherein said step of maintaining said store further comprises the steps of:

maintaining a first store that contains first values contained in said resource during accesses thereof by said first simulated process; and

maintaining a second store that contains second values contained in said resource by said second simulated process during accesses thereof, wherein said first values comprise first read values and first written values, and said second values comprise second read values and second written values; and

said step of determining whether said store contains non-unique values further comprises the step of identifying in one of said first store and said second store a last value written to said

resource in said step of simulating said one of said generated instructions.

39. (canceled).

40. (original) The computer software product according to claim 27, further comprising the step of establishing a synchronization barrier for said first simulated process and said second simulated process.

41. (original) The computer software product according to claim 27, further comprising the step of biasing generation of said test program to promote collisions of memory accessing instructions that are executed by said first simulated process and said second simulated process.

42. (currently amended) A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to perform a method of verification of a multiprocessor architecture by simulation, comprising the steps of:

defining a program input to a test generator;

generating a multi-processor architecture test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results comprising a plurality of mutually dependent non-adjacent resources having non-contiguous addresses, wherein at least one member of said list of predicted resource results comprises a value-list containing permissible

values of one of said mutually dependent non-adjacent resources ~~a resource~~;

initializing said mutually dependent non-adjacent resources;

using said initialized mutually dependent non-adjacent resources simulating an execution of said test program using a plurality of simultaneously executing processes;

creating, by the computer executing said test program, a set of tagged value-lists, each tagged value-list comprising said value-list of said at least one member and identifying an allowed subcombination of values of one of the mutually dependent non-adjacent resources resulting from execution said test program, and formed by tagging members of a list of predicted results with a combination identifier which includes a string of literals identifying a particular outcome of said test program, wherein at least a portion of said mutually dependent non-adjacent resources have a common combination identifier, each of said tagged value-lists further comprising said combination identifier;

upon completion of an execution of said processes comparing contents of said mutually dependent non-adjacent resources with the correspondingly tagged member of said set of tagged value-lists to match the allowed subcombination of values thereof, said step of comparing performed by selecting commonly tagged members of said set of tagged value-lists for each said combination identifier; and

responsively to a failure to match any allowed subcombination of values in iterations of said step of comparing reporting that an architectural violation of the multi-processor architecture exists

~~creating, by the computer executing said test program, a tagged value-list by tagging members of a list of predicted results with a combination identifier which includes a string of literals~~

~~identifying a particular outcome of said test program, said tagged value-list comprising:~~

~~a set of non-unique values; and~~

~~said combination identifier replacing said value-list by said tagged value-list, such that said set of tagged value lists is used in place of said set of value-lists; and~~

~~verifying the architecture if a content of said resource is equal to a member of said tagged value-list~~.

43. (previously presented) The computer software product according to claim 42, wherein said list of predicted resource results comprises predicted results of adjacent resources having contiguous addresses, wherein said adjacent resources are mutually dependent, and said step of verifying said actual resource result is performed by verifying each of said adjacent resources.

44. (original) The computer software product according to claim 43, wherein said adjacent resources are memory resources.

45. (original) The computer software product according to claim 43, wherein said adjacent resources are registers.

46. (original) The computer software product according to claim 42, further comprising the step of establishing a synchronization barrier for said simultaneously executing processes.

47. (original) The computer software product according to claim 42, further comprising the steps of biasing generation of said test program to promote collisions of memory accessing

instructions that are executed by said simultaneously executing processes.

48. (previously presented) The computer software product according to claim 42, further comprising the steps of:

identifying a combination of said mutually dependent non-adjacent resources by tagging corresponding members of said list of predicted resource results with a unique combination identifier of a particular outcome of said test program so as to define commonly tagged lists of values of predicted resource results; and

wherein said step of verifying said actual resource result is performed by verifying that resources of said combination have actual results that are equal to a member of a corresponding one of said commonly tagged lists of values.

49. (currently amended) A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to perform a method of predicting non-unique results by simulating a system design, comprising the steps of:

defining a program input to a test generator;

generating a multi-processor architecture test program responsive to said program input, said test program including a list of resource initializations, a list of instructions, and a list of predicted resource results comprising a plurality of mutually dependent non-adjacent resources having non-contiguous addresses, wherein at least one member of said list of predicted resource results comprises a value-list containing permissible values of a resource;

initializing said mutually dependent non-adjacent resources;

using said initialized mutually dependent non-adjacent resources simulating an execution of a single instruction of said test program by a first process of said test program;

calculating possible values of target resources of said single instruction;

creating, by the computer executing said test program, a set of tagged ~~value-list~~ value-lists by tagging members of a list of predicted results with a combination identifier which includes a string of literals identifying a particular outcome of said test program, each said tagged value-list identifying an allowed subcombination of values of one of the mutually dependent non-adjacent resources, wherein at least a portion of said mutually dependent non-adjacent resources have a common combination identifier, comprising:

~~a set of non-unique~~ a list of said permissible values; and

said combination identifier; and

~~replacing said value-list by said tagged value-list, such that said set of tagged value-lists is used in place of said set of value-lists;~~

upon completion of an execution of said single instruction comparing contents of said mutually dependent non-adjacent resources with the correspondingly tagged member of said set of tagged value-lists to match the allowed subcombination of values thereof, said step of comparing performed by selecting commonly tagged members of said set of tagged value-lists for each said combination identifier; and

responsively to a failure to match any allowed subcombination of values in iterations of said step of comparing reporting that an architectural violation of the multi-processor architecture exists.

50. (previously presented) The computer software product according to claim 49, the method further comprising the steps of:

performing said step of simulating an execution by a second process of said test program;

maintaining lists of written values that are written to said target resources of said single instruction by said first process and said second process; and

determining respective last values in said lists of written values.

51. (previously presented) The computer software product according to claim 50, the method further comprising the steps of:

prior to performing said steps of simulating said execution by said first process and of simulating said execution by said second process memorizing an initial simulated state of said test program;

maintaining lists of read values that are read from source resources of said single instruction;

identifying a member of said lists of read values so as to define an identified member;

restoring said initial simulated state of said test program; and

performing said step of simulating said execution a second time, and reading said identified member, using an associated process other than the first and second processes.

52-70. (canceled).

71. (previously presented) The computer software product according to claim 27, wherein said mutually dependent non-adjacent resources comprise resources of different types.

72. (previously presented) The computer software product according to claim 71, wherein said different types comprise memories and registers.

73. (previously presented) The computer software product according to claim 42, wherein said non-adjacent resources are memory resources.

74. (previously presented) The computer software product according to claim 42, wherein said non-adjacent resources are registers, and wherein the addresses comprise indexes.

75. (previously presented) The computer software product according to claim 42, wherein said mutually dependent non-adjacent resources comprise resources of different types.

76. (previously presented) The computer software product according to claim 75, wherein said different types comprise memories and registers.

77. (previously presented) The computer software product according to claim 49, wherein said mutually dependent non-adjacent resources comprise resources of different types.

78. (previously presented) The computer software product according to claim 77, wherein said different types comprise memories and registers.